[Ingalls78a] D. H. Ingalls, "The Smalltalk-76 Programming System Design and Implementation," pp. 9-15 dans *Conference Record of the 5th Annual ACM Symposium on Principles of Programming Languages*, Tucson (January 1978).

[Jakob84a] F. Jakob, "Métaclasses: en avoir ou pas?," *Bigre*, (22-23 Novembre 1984). 2 Journées d'étude sur les L.O.O

[Kay69a] A. Kay, *The Reactive Engine*, Ph. D. Thesis 1969.

[Kay76a] A. Kay and A. Goldberg, "SMALLTALK-72 Instruction Manual," SSL 76-6, Xerox Palo Alto Research Center, Palo Alto, CA. (March 1976).

[Lieberman81a] H. Lieberman, "A Preview of Act1," AI Memo No. 625, MIT (June 1981).

[Meyer79a] B. Meyer, "Les concepts de SIMULA 67," Atelier Logiciel, EDF, Clamart 92141 (Juin 1979).

[Moon80a] D. A. Moon and D. Weinreb, "Flavors: Message Passing In The Lisp Machine," AI Memo No 602, MIT (November 1980).

[OBJET-AFCET84a] Grp OBJET-AFCET and Brest, "Actes des deuxièmes journées d'étude du groupe de travail AFCET sur les LANGAGES ORIENTES OBJETS," dans *BIGRE + GLOBULE*, ed. J. Bezivin & P. Cointe, Brest (22-23 Novembre 1984).

[Queinnec84a] C. Queinnec, *LISP Mode d'emploi*, Eyrolles, Paris (1984).

[Rees84a] J. A. Rees, N. I. Adams, and J. R. Meehan, *The T Manual (Fourth edition)*. January 1984.

[Rodet84a] X. Rodet and P. Cointe, "Formes: Composition and Scheduling of Processes," *Computer Music Journal* 8(3) p. 32 50 MIT Press, (Fall 1984).

[Scherer72a] C. Scherer, "SIMULA par l'exemple," 5224 T/Fr , C.I.I Louveciennes (Décembre 1972).

[Serpette85a] B.P. Serpette, "Parallélisme et langages applicatifs," *Bigre+Globule (journées GROPLAN-C3-GRECO)*, (Avril 1985).

[Serpette84a] B.P. Serpette, "Contextes, Processus, Objets, Séquenceurs: FORMES.," LITP 85-5, Université Paris VI, Paris (Octobre 84). Thèse de 3ème cycle

[Steele84a] G. L. Steele, *COMMON LISP (The language)*, Digital Press (DEC) (1984).

[Yonezawa85a] A. Yonezawa and Y. Matsumoto, *Object Oriented Programming and Industrial Software Production*, TAPSOFT, Berlin (March 85).

LES METACLASSES DANS LES LANGAGES ORIENTES OBJETS

METACLASSES IN OBJECT ORIENTED LANGUAGES

Jean-Pierre BRIOT (*)

RÉSUMÉ - Une des caractéristiques principales des Langages Orientés Objets est la volonté d'uniformité de leur univers. Le statut des classes en SMALLTALK n'est cependant pas conforme à cet idéal. Le concept de métaclasse qu'introduit SM*LLTALK 80 pour tenter de redonner un statut d'objet à part entière aux classes complique inutilement le modèle, sans en résoudre les limites. Nous en proposons donc une simplification, permettant à la fois une clarification et un plus grand pouvoir expressif. Nous évoquerons des exemples d'application dans divers domaines, puis un parallèle avec le modèle des acteurs où ce problème de statut ne se pose pas.

Mots clés : objet - classe - métaclasse - instance - uniformité - instanciation - variables de classe - SM*LLTALK 80 - ObjVlisp - acteur

ABSTRACT - One of the principal characteristics of Object Oriented Languages is the uniformity of their universe. The classes of SMALLTALK, however, do not conform to this ideal. The metaclass concept introduced by SM*LLTALK 80 attempts to elevate classes to the level of objects but tends to complicate the model without resolving the limitations. We will propose a simplification designed to permit greater clarity and expressive power. We will describe examples in various domains of application, and a parallel with the actor metaphor where this problem of level does not arise.

Keywords : object - class - metaclass - instance - uniformity - instanciation - class variables - SM*LLTALK 80 - ObjVlisp - actor

* IRCAM - 31, rue Saint-Merri 75004 PARIS
LITP PARIS VI - 5, place Jussieu 75005 PARIS
ircam!jpb - litp!jpb - geocub!jpb

1. INTRODUCTION

Object Oriented Programming is a phrase that is beginning to catch on, just like the phrase *structured programming* did in the '70s [Horowitz83].

La Programmation Orientée Objet est en train de devenir un standard, en témoigne la création d'un groupe "Langages Orientés Objets" à l'AFCET. Rapidité de conception, modularité, extensibilité et fiabilité sont ses principaux avantages. Ses domaines d'application sont chaque jour plus nombreux, notamment en représentation de connaissances avec les langages MERING [Ferber85], LRO [Roche85], KOOL [Albert83], LOOPS [Bobrow83], GLISP [Novak83] ... héritiers de CONNIVER [McDermott72] et de KRL [Bobrow77]. Enfin une retombée grand-public de ces idées est le déjà célèbre Mac Intosh, héritier spirituel du *dyna-book* défini par Alan Kay au début des années 70 et dont le langage s'appelait SMALLTALK [Kay69].
La qualité première du modèle objet est sa simplicité. Il repose en effet sur une dualité entre l'*objet*, unique entité de l'univers, résultant de l'unification des données et des procédures [Cointe83], et le *message*, moyen de communication entre objets, unique structure de contrôle.
Cette simplicité et clarté du modèle est fondamentale surtout lors du développement de systèmes de très haut niveau, particulièrement en Intelligence Artificielle.
Or le modèle objet tel qu'il a été établi par SMALLTALK [Kay76] ne respecte pas cette volonté pourtant déclarée d'uniformité. Les classes ne sont en effet pas des objets comme les autres. Les métaclasses proposées par SM*LLTALK 80 [Goldberg83] ne sont qu'un pis aller et compliquent inutilement le modèle. Ce problème est bien réel et a déjà été soulevé dans la lettre des utilisateurs de SMALLTALK [Doyle84].
Nous nous proposons ici d'expliquer tout d'abord en quoi ce modèle des métaclasses n'est pas satisfaisant et pourquoi les choix d'implémentation non remis en cause imposent cet état de fait. Puis nous proposerons un nouveau modèle de classes et de métaclasses à la fois plus clair, plus général et plus puissant. Nous décrirons son implémentation et ses applications, notamment au développement du système FORMES [Cointe84, Cointe84a, Rodet84] destiné à la Composition et à la Synthèse Musicale par Ordinateur. Nous terminerons par un bref parallèle avec le modèle *acteur*.

2. CLASSE ET METACLASSE

2.1 Objet et classe

Le modèle *objet* est basé sur le concept de *classe* initialement introduit par SIMULA 67 [Birtwistle73], puis repris et systématisé par Alan Kay et son groupe dans le langage SMALLTALK [Kay76]. Une classe SIMULA est un moyen de décrire un type abstrait, ou plus exactement la structure de données associée au type qu'elle représentera.
Des exemplaires de cette structure seront ensuite créés dynamiquement par l'utilisateur pendant l'exécution du programme à l'aide de l'opérateur *new* sur la classe, ce seront les *objets*, *instances* de la classe.
La classe est donc l'analogue du moule, apte à générer des pièces, exemplaires de la structure qu'elle incarne.

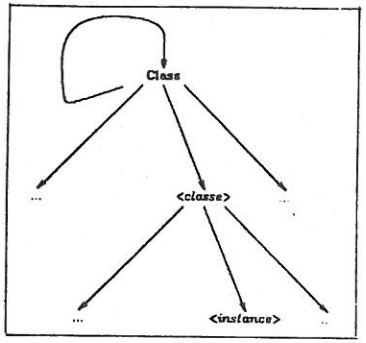
2.2 Uniformité

En SMALLTALK, l'interactivité et la dynamicité du langage permet une déclaration à tout moment. L'équivalence entre données et programme (de même qu'en LISP) tend à

conférer alors à la classe le statut d'objet, ce qui va entraîner un des énoncés fondamentaux de SMALLTALK formulé par Alan Kay, lui même:

Every object belongs to a class.

SMALLTALK a adopté une attitude d'uniformité quand à son univers en considérant l'objet comme son *unique* entité.
Cette doctrine radicale n'est pas pour autant appliquée jusqu'au bout quand à la nature des classes dans la première version du langage (SM*LLTALK 72 [Kay76]). Une classe n'est pas un objet comme les autres, n'étant pas une instance créée par instanciation d'une classe, mais par la primitive *to*.
Cette incomplétude sémantique disparaît dans SM*LLTALK 76 [Ingalls78], une classe devenant elle même instance de la classe *Class*.
En reprenant notre analogie, nous dirons donc qu'un moule passe maintenant au rang de pièce, et est donc lui-même issu d'un moule, la classe *Class* que l'on appellera à ce titre *métaclasse*.
On voit alors surgir les dangers du raisonnement régressif.
SM*LLTALK 76 arrête cette régression infinie en introduisant une circularité en haut de l'arbre d'instanciation, i.e. *Class* est instance (ou classe) d'elle même.
Voici l'arbre d'instanciation [Cointe83, Briot83] de SM*LLTALK 76:

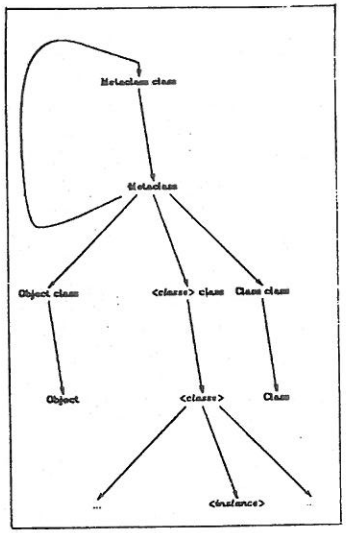


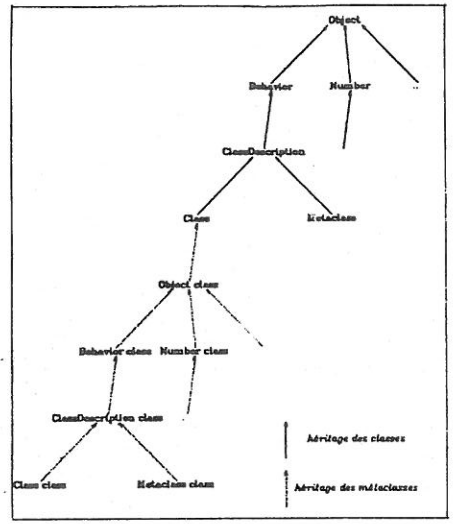
L'existence d'une unique métaclasse (*Class*) impose cependant un comportement commun à l'ensemble des classes.
Pour remédier à cet inconvénient, SM*LLTALK 80 [Goldberg83, Althoff81] amène une généralisation de la métaclasse *Class*.

3. LES METACLASSES SM*LLTALK 80

3.1 Métaclasses et méthodes de classe

Chaque classe possède maintenant sa propre métaclasse, permettant ainsi de spécifier des comportements particuliers pour une classe donnée.
La métaclasse sera principalement utilisée pour modifier ou construire de nouvelles méthodes d'instanciation et d'initialisation. Ces méthodes opérant sur la classe elle même, sont déclarées à sa création comme *méthodes de classe* (par opposition aux *méthodes d'instances*).
Les deux arbres d'instanciation et d'héritage deviennent maintenant passablement complexes, comme le montrent les deux figures, page suivante:





3.2 Critique

De plus malgré les apparences, le modèle original est toujours préservé. En effet les métaclasses proposées sont implicites et virtuelles:

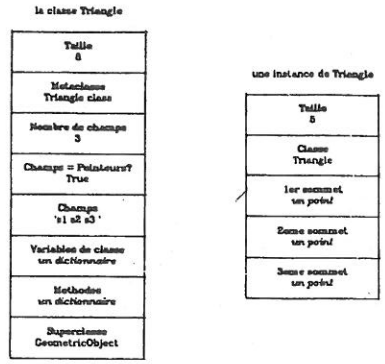
- *implicites*,
  chaque métaclasse n'existe qu'à travers la spécification des méthodes de classe et n'est pas créée explicitement en tant que classe comme les autres par l'utilisateur.

- *virtuelles*,
  une métaclasse n'est pas réellement accessible à l'utilisateur, par exemple on ne peut pas l'instancier, son unique instance étant la classe qui l'aura induite.

Ces métaclasses bénéficient également d'une hiérarchie parallèle à leurs classes, comme on le voit dans l'arbre d'héritage, mais elles n'ont néanmoins pas le statut de classes comme les autres et ne sont pas directement accessibles à l'utilisateur.
Une classe n'a donc toujours pas un statut d'instance à part entière, son générateur (métaclasse) étant créé à partir d'elle et non l'inverse.

3.3 Pourquoi?

Une classe SMALLTALK est clairement une mise en commun des instances qu'elle génère. Le *dictionnaire des méthodes* et la moitié du *dictionnaire des champs* (constituée des variables d'instances) la constitue. Voici par exemple, page suivante, la représentation interne de la classe Triangle et d'une de ses instances, exemple tiré de [Althoff81].



On remarque qu'une classe SM*LLTALK 80 ne possède pas de champs propres, exception faite des *variables de classe*, tentative pour leur assurer un environnement mais qui n'est en fait qu'un ensemble de variables partagées par un ensemble de classes (*pool variables*) et qui n'est pas héritable.
Une classe n'est donc pas un objet comme les autres, ne détenant aucune valeur. Le modèle SMALLTALK reste ainsi prisonnier de ses choix d'implémentation.

4. NOUVEAU MODELE

Nous allons maintenant proposer un nouveau modèle simplificateur et aux possibilités plus étendues, mais auparavant récapitulons la structure de l'univers des objets.

4.1 Instance, classe et métaclasse

On considère au départ deux catégories d'objets:

- les générateurs, ou *classes*,
  définissant un type de données et ayant le pouvoir de générer des instances.

- et les *instances terminales*,[1]
  n'ayant pas ce pouvoir.

Cette distinction rebondit comme on l'a vu sur les classes, et par conséquent la première catégorie se subdivise elle même en deux types:

- les *métaclasses*,
  générateurs d'objets non terminaux, i.e. des classes.

- et les *classes simples*,
  générant des instances terminales.

Un générateur de métaclasses est, par inclusion du sous-ensemble des métaclasses dans l'ensemble des classes, un générateur de classes, autrement dit une métaclasse. La division de l'univers des objets s'en arrête donc là sur ces trois sous-ensembles.

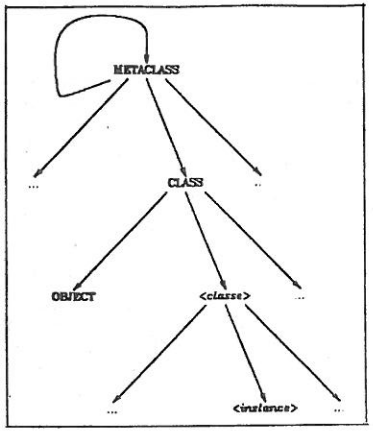
1. souvent appelées *instances* par abus de langage.

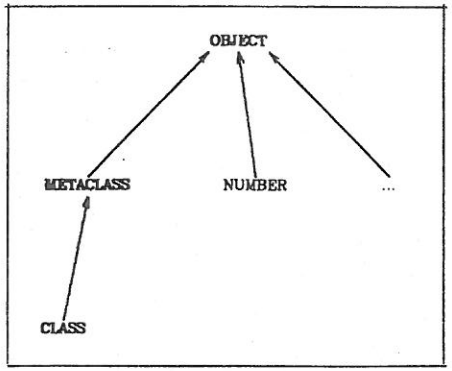
Aux deux derniers types correspondent deux sémantiques différentes du message d'instanciation *new* (générer une classe ou bien générer une instance terminale).
Il est donc souhaitable d'incarner ces deux méthodes distinctes dans deux objets distincts.
Nous pouvons maintenant exposer les bases de ce nouveau modèle:

4.2 Nouveau modèle

Nous nommerons METACLASS le générateur du premier des types, et CLASS celui du deuxième.
CLASS est donc instance de METACLASS, cette dernière étant son propre générateur.

4.3 Arbres d'instanciation et d'héritage





4.4 Implémentation

Ce nouveau modèle a été initialement implémenté en ObjVlisp [Cointe84b, Cointe84c]. Il utilise les primitives (certaines sont légèrement modifiées) de la machine virtuelle présentée dans [Cointe85], et qui ne seront donc pas détaillées ici.
L'*astuce* évitant une régression infinie se traduit au niveau de l'implémentation par une amorce (*boot-strap*) du germe de METACLASS qui pourra ainsi s'engendrer elle même. Ce germe minimal est réduit à sa structure et à la méthode *new* génératrice de classes.

```
(set-fields 'OBJECT ())          ; pour l'héritage de la liste (vide) des champs de OBJECT

(make-class 'METACLASS () 'METACLASS '(OBJECT) () ())

(newmethod 'METACLASS 'new
  (lambda (name supers fields . methods)
    (make-class oself (fields oself) name supers fields methods)))
```

Nous pouvons maintenant générer METACLASS à partir d'elle même, comme sous-classe de OBJECT qui sera définie ensuite.


```
(send 'METACLASS 'new 'METACLASS '(subclassof: OBJECT) '(fields:)
  'new          (methodfor 'METACLASS 'new)
  'fields       (λ () (fields oself))
  'subclassof   (λ () (supers oself))
  'selectors    (λ () (selectors (m-dico oself)))
  'understands  (λ (sel method) (defmethod oself sel method))
  'derstands    (λ (sel) (killmethod oself sel))
  'methodfor    (λ (sel) (methodfor oself sel)) )
```

METACLASS va ensuite s'instancier pour donner naissance à CLASS détentrice de la méthode *new* générant des instances terminales.

```
(send 'METACLASS 'new 'CLASS '(subclassof: METACLASS) '(fields:)
  'new      (λ (name)
              (make-object oself name (oself 'fields)))
  'new:     (λ (name . values) (send (oself 'new name) 'values values) name) )
```

Viendrait ensuite la création de la classe OBJECT, racine de la hiérarchie.

Tout objet possède maintenant des valeurs associées aux variables d'instances détenues par sa classe. On lui donne de plus un accès non seulement à son environnement propre, mais de plus à l'environnement de sa classe. Ceci permet de simuler ainsi de manière tout à fait naturelle le concept de variables de classes, avec des possibilités de niveaux illimités et d'héritage des environnements, ce qui est impossible en SM*LLTALK 80.

L'implémentation du modèle dans son intégralité ainsi que son utilisation dans divers contextes se trouve dans [Briot84].

4.5 Avantages et possibilités

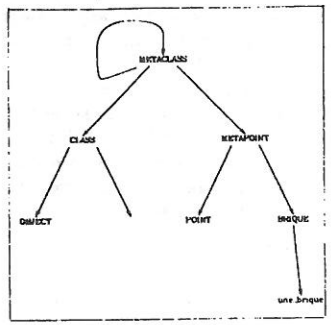
Le premier avantage de ce nouveau modèle est sa simplicité, il permet d'éviter le recours à des notions supplémentaires telles les méthodes de classe et les variables de classe. Cette simplification est extrêmement importante, elle permet de retrouver l'uniformité au coeur du modèle objet, et en réduisant le nombre de concepts, augmente leur clarté.
Le second avantage est la généralité de ce modèle de classes, et leur puissance d'expression non limitée. On peut désormais augmenter sans limitation la profondeur de l'arbre d'instanciation pour exprimer divers concepts et niveaux de regroupements de connaissances et d'actions.
Les méthodes de classe et variables de classe deviennent maintenant des méthodes et variables à part entière, détenues par la métaclasse. Ainsi tous les niveaux sont totalement accessibles, et ce de manière uniforme, à l'utilisateur.
Signalons qu'un modèle de métaclasses explicites est également présenté dans LOOPS [Bobrow83], mais leur modèle possède une profondeur limitée et un manque d'uniformité comme pour SM*LLTALK 80, et de ce fait des possibilités semblables.

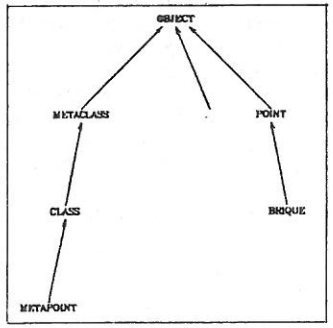
4.6 Exemple

Pour mieux illustrer ces nouvelles possibilités, prenons un exemple très simple. Pour la construction d'un jeu vidéo (où de petits animaux tentent de sortir de labyrinthes) sur des terminaux alphanumériques, le premier pas est la création d'une classe de points sur l'écran. Le caractère associé (la variable *car*) correspond à l'objet visualisé, ainsi par exemple le "=" pour les briques avec lesquelles seront construits les murs du labyrinthe. Il est donc déclaré dans la métaclasse de la classe POINT, appelons la METAPOINT.

```
? (send 'METACLASS 'new 'METAPOINT '(subclassof: CLASS) '(fields: car))
= METAPOINT                                  ; on crée une métaclasse
?
? (send 'METAPOINT 'new 'POINT '(subclassof: OBJECT) '(fields: x y)
?  'affiche    (λ () (tycursor x y) (tyflush) (prinflush car)))
= POINT                                      ; on l'instancie en la classe POINT
? (send 'POINT 'values: "*")                 ; le caractère associé est le "*"
= (*)
?
? (send (send 'METAPOINT 'new 'BRIQUE '(subclassof: POINT) '(fields:))
?  'values: "=")                             ; on l'instancie en une nouvelle classe
= (=)                                        ; le caractère est cette fois le "="
?
? (send 'BRIQUE 'new: 'une-brique 20 20)     ; on crée une brique,
= une-brique                                 ; instance terminale de la classe BRIQUE
?
? (send 'une-brique 'affiche)                ; elle s'affiche sur l'écran
                                =
= =
```

La classe POINT possède la valeur "*" associée à la variable *car*, et déclare les variables *x* *y*. Ainsi toutes les instances de POINT s'afficheront par des "*" sur l'écran, et les briques par des "=". On pourrait de même créer la classe des oubliettes, représentées par le caractère "o", et d'autres classes d'objets sur l'écran.
Voici les deux arbres correspondant à cette construction très simple. Elle est pourtant impossible en SM*LLTALK 80, les variables de classe n'étant pas héritables, le caractère serait alors le même pour toutes les sous-classes de POINT.





La suite de l'exemple, qui introduit une simulation des ensembles par les classes (un labyrinthe est un ensemble de briques), est exposée dans [Briot84]. Ce modèle permet de construire très naturellement les divers niveaux de classes alors nécessaires.

4.7 Applications

Ce nouveau modèle a déjà été utilisé en ObjVlisp [Cointe84c]. Il a de plus été choisi pour le développement d'un nouveau dialecte de SMALLTALK dans le cadre du projet META [Briot84a] entre l'Université PARIS VI et le CNDP et destiné aux écoles et aux universités. Il est également utilisé dans le développement actuel du langage FORMES [Briot84].

5. ET LES ACTEURS?

Le modèle des acteurs développé par Carl Hewitt et son équipe dans les langages PLASMA [Hewitt75], puis ACT1 [Lieberman81] n'est pas concerné par ce concept de métaclasse. Le concept de classe est en effet absent du modèle acteur. Le modèle d'instanciation est cette fois différent, c'est un modèle de *copie différentielle* [Barthes53, Cointe84c]. Dans ce modèle, toute entité est un générateur, à la différence du modèle objet. La problématique précédente ne se pose donc pas. On pourra voir une application de cet autre modèle dans [Jakob84], description d'un système d'aide à la conduite de processus industriels .

6. CONCLUSION

Ce nouveau modèle simplificateur, ne se limite pas au seul langage SMALLTALK dont il est issu. Il est déjà utilisé dans une grande variété d'applications. Il n'est pas seulement unificateur et réducteur du nombre de concepts, mais également plus général et de ce fait plus puissant et manipulable. Un gain de clarté n'est pas un luxe mais une nécessité dans des utilisations de haut niveau.

7. REFERENCES

[Albert83] P. Albert, "KOOL: représentation des connaissances ," *Bigre*, (37) pp. 88-91 (Octobre 1983). Premières Journées L.O.O.

[Althoff81] J. C. Althoff, L. P. Deutsch, D. H. Ingalls, A. Goldberg, T. Kaehler, G. Krasner, F. H. Reenskaug, D. Robson, J. Ross, et L. Tesler, "The Smalltalk-80 System," Learning Research Group, BYTE Vol6-No 8, Mc Graw-Hill Publication (August 1981).

[Barthes53] R. Barthes, "Le degré zéro de l'écriture," pp. 100 dans *Bibliothèque Méditations*, ed. Editions Gonthier, (1953).

[Birtwistle73] G. Birtwistle, O. Dahl, B Myhrhaug, et K Nygaard, *SIMULA BEGIN*, Petrocelli/Charter, New York (1973).

[Bobrow83] D. Bobrow et M Stefik, *The LOOPS Manual*. December 83.

[Bobrow77] D. G Bobrow et T. Winograd, *An Overview of KRL A Knowledge Representation Language*, Vol. 1, Cognitive Science (1977).

[Briot84a] J.P. Briot et E. Saint-James, "Intégration de SMALLTALK dans le projet AGLAE.," *Bigre+Globule*, (41) pp. 52-72 (22-23 Novembre 1984). 2e Journées d'etude sur les L.O.O.

[Briot83] J.P. Briot, "L'instanciation dans les langages objets," *Bigre*, (37) pp. 173-209 (Décembre 83). Premières Journées L.O.O.

[Briot84] J.P. Briot, "Instanciation et Héritage dans les Langages Objets.," Rapport LITP 85-21, Université Paris-6, Paris (Décembre 84). Thèse de 3ième cycle

[Cointe84b] P. Cointe, "Une extension de Vlisp par les Objets," *Science of Computer Programming*, (161)North Holland, (1984).

[Cointe85] P. Cointe, *Le modèle OBJVLISP une plate-forme pour l'expérimentation des formalismes Objets*, 5ème congrès AFCET-RFIA , Grenoble (25-27 Novembre 1985).

[Cointe83] P. Cointe, "Penser Smalltalk, Comprendre Smalltalk," *Bigre*, (37) pp. 13-54 Journée d'étude sur les langages orientés objet, (Décembre 83).

[Cointe84] P. Cointe et X. Rodet, "FORMES: un langage-objet gérant des processus hiérarchisés," pp. 291-311 dans *Quatrième Congrès Reconnaissance des Formes et Intelligence Artificielle*, Vol. 2, INRIA AFCET, Paris (25-27 Janvier 1984).

[Cointe84a] P. Cointe et X. Rodet, *Formes: an Object & Time Oriented System for Music Composition and Synthesis*, Conference Record of the 1984 ACM symposium on LISP and Functional Programming, Austin (Texas) (5-8 August 84).

[Cointe84c] P. Cointe, "Implémentation et interprétation des langages objets, application aux langages Formes, ObjVlisp et Smalltalk (thèse d'Etat).," LITP 85.?, Université Paris-VI, IRCAM (17 Décembre 84).

[Doyle84] P. Doyle, "Letter to the Editor How about Protoclass?," *SMALLTALK-80 Newsletter*, (4)(September 1984).

[Ferber85] J. Ferber, "MERING II: Langage et Métalangage pour la création de formalismes orientés objets.," dans *Matériels et Logiciels pour la 5ème génération*, AFCET, Paris (5,7 Mars 1985).

[Goldberg83] A. Goldberg et D. Robson, *SMALLTALK-80 The Language and its Implementation*, Addison-Wesley Publishing Compagny (1983).

[Hewitt75] C. E. Hewitt et B. Smith, *A PLASMA Primer*, MIT Artificial Intelligence Laboratory (September 1975). Draft

[Horowitz83] E. Horowitz, *Fundamentals of Programming Languages*, Springer-Verlag, Berlin-Heidelberg New York (1983).

[Ingalls78] D. H. Ingalls, "The Smalltalk-76 Programming System Design and Implementation," pp. 9-15 dans *Conference Record of the 5th Annual ACM Symposium on Principles of Programming Languages*, Tucson (January 1978).

[Jakob84] F. Jakob, "Métaclasses, en avoir ou pas?," *Bigre+Globule*, (41 2e Volume) pp. 5-14 (22-23 Novembre 1984). 2e Journées d'etude sur les L.O.O.

[Kay69] A. Kay, *The Reactive Engine*, Ph. D. Thesis 1969.

[Kay76] A. Kay et A. Goldberg, "SMALLTALK-72 Instruction Manual," SSL 76-6, Xerox Palo Alto Research Center, Palo Alto, CA. (March 1976).

[Lieberman81] H. Lieberman, "A Preview of Act1," AI Memo No. 625, MIT (June 1981).

[McDermott72] D. V. McDermott et G. J. Sussman, "The Conniver Reference Manual," 259a, MIT Memo (May 72). Updated January 74

[Novak83] G. S. Jr Novak, "GLISP: A Lisp-based Programming System with Data Abstraction," *The AI Magazine*, (Fall 1983).

[Roche85] C. Roche et J.P. Laurent, "EAQUE - LRO: un outil pour la génération de systèmes experts," dans *Matériels et Logiciels pour la 5ème génération*, AFCET, Paris (5,7 Mars 1985).

[Rodet84] X. Rodet, P. Cointe, J.B. Barrière, Y. Potard, B. Serpette, et J.P. Briot, *Démonstration de FORMES et son utilisation pour contrôler des synthétiseurs*, International Computer Music Conference, IRCAM (19-23 Octobre 1984).

CHAPITRE 13

ASPECTS FONDAMENTAUX DE LA RECONNAISSANCE DES FORMES (2)